Transport Polynomials

Software for "Learning Nonlinear Continuous-Time Systems for Formal Uncertainty Propagation and Probabilistic Evaluation"

Requirements

- Docker

Expected Outcomes

The proposed method is data-driven, and in our case studies, the data is generated randomly. This means that there may be small variations between executions, however, we found that the variations are almost negligible, and the outcomes are very consistent. Additionally, the comparison Monte Carlo methods rely on randomness, so the results are subject to variation trial-by-trial, but the trend should be consistent.

How to use

1. Build and start a docker container

```
docker compose up --build -d
```

2. Exec into the container

```
docker exec -it transportpolynomialsjl-ws-1 /bin/bash
```

3. Once inside the container, the scripts can be run using

```
julia scripts/<script name>.jl
```

For example,

```
julia scripts/van_der_pol_2D.jl
```

If everything ran correctly, you should see terminal output `Plots saved to figures directory`. 4) The results are saved in the `TransportPolynomials.jl/figures_container/` directory (outside the container) and the `app/figures/` directory inside the container. Those two directories are mounted, so the files are shared. Once a script has finished running, it will put all generated results figures in that directory.

Paper Outcomes and Artifact Outputs

- Figure 2.a.1 and 2.a.2 are generated by `van_der_pol_2D.jl`
- Figure 2.b.1 and 2.b.2 are generated by `van_der_pol_2D_rare_event.jl`
- Figure 3.a is generated by `cartpole_4D.jl`
- Figure 3.b is generated by `cartpole_4D_rare_event.jl`
- Figure 4 is generated by `van_der_pol_2D_ablation_m.jl`

Modification of a Script/Using the Tool

Referring to the `van_der_pol_2D.jl` script as an example, the specifications at the top of the file can be modified for a given application as follows:

Create the true underlying system to sample data from (see `src/Systems.jl` for example systems and a reference on how to create your own):

```
true_system, dtf = van_der_pol(μ=1.0)
```

Create the region of integration

```
target_region = Hyperrectangle(low=[0.0, 0.0], high=[0.3, 0.3])
```

Specify the desired time instance

```
duration = 5.0
```

Specify the polynomial regression degrees for each component of the vector field

```
model_degrees = 7 * ones(Int, dimension(true_system))
```

Specify the degree of the flow pipe expansions

```
fp_deg = 4
```

Specify the degree of the volume function taylor expansion

```
vp_deg = 5
```

Specify the flowpipe box time interval between boxes (fixed intervals for this implementation)

```
Δt_max = 0.1
```

Add regularization to the polynomial regression

```
λ = 10.0
```

Save the plots to the figures directory

```
save_plots = true
```